14.5

University of Bahrain
College of Information Technology
Department of Computer Science
Summer Semester, 2012 – 13

ITCS215 Data Structures

Quiz 1

Name **ID #** **Sec**

The following class is to store a bank customer's account number and balance.

```
class bankAccount
{
 public:
   void setNumber(int num);
   void setBalance(double num);
   int getNumber();
   double getBalance();
   void deposit (double d);
   void withdraw(double w);
   void print();//print the account number and the balance
   bankAccount(int n, double b);
 private:
   int number; // account number
   double balance;
};
```

Derive a class **checkingAccount** from the class **bankAccount** (public inheritance) with the following members:

(a) **private** data members:
- interest (double)
- minimumBalance (double)
- serviceCharges (double)

(b) **public** member functions:
- functions to set the interest, the minimum balance, and the service charges.
- functions to get the interest, the minimum balance, and the service charges.
- checkBalance() to apply service charges (new balance will become old balance – service charges), if the balance falls below the minimum balance and return true, otherwise returns false.
- print() to print the account information (including the information of the base class bankAccount).
- A constructor with 5 parameters.

Implement all the member functions of the class **checkingAccount**.

(14)

University of Bahrain
College of Information Technology
Department of Computer Science
Summer Semester, 2012 – 13

ITCS215 Data Structures

Quiz 2

Name **ID #** **Sec**

(1) [7 Marks] Write the member function **`insertAt`** of class **`arrayListType`** as discussed in the lectures.

(2) [8Marks] Write a non-member function called **`merge`** that accepts two array-based lists *list1* and *list2* of type **`arrayListType`** as parameters. Elements of *list1* are sorted in ascending order. Whereas, the elements of *list2* are not sorted. The function should insert the elements of *list2* in *list1* in such a manner that after insertion also *list1* remains sorted. Assume that class **`arrayListType`** is available for use.

Function prototype:

```
template<class elemType>
bool merge(arrayListType<elemType> &list1,
           const arrayListType<elemType> &list2);
```

Note that the function returns *false*, if both lists *list1* and *list2* are empty or if *list1* becomes full at any stage, else returns *true*.

```
template <class type>
void arrayListType::insertAt(int loc, const type &item)
{
  if (isFull()){cerr<<"List is Full "<<endl;}

  else { for (int i=length; i>loc ; i--)
           list[i]=list[i-1];
         list[loc]=item;}
  length++;
}
```

(6)

(13)

University of Bahrain
College of Information Technology
Department of Computer Science
Summer Semester, 2012 – 13

ITCS215 Data Structures

Quiz 3

Name **ID #** **Sec**

Write a function ***insertAt*** to be included as a member function in class ***linkedListType***, to insert a data value *newItem* at index *position*. Index *position* and data value *newItem* are passed as parameters to the function. The index of the first node is 0 and increases by 1 for each subsequent node. If the index is invalid then the function does not insert *newItem* and returns *false*. Else, the function returns *true* after insertion.

Function prototype:

```
bool insertAt(int position, const Type& newItem);
```

Example:

Before insertion

list: 5 10 20 15 22 33

position: 2 newItem: 30

After insertion

list: 5 10 **30** 20 15 22 33

```
template <class type>
bool linkedListType<type>::insertAt(int pos, const Type & item)
{ if (first == NULL)
  { cerr<<"List is Empty"<<endl; return false;}
  if( pos <0 || pos > count)
  { cerr<<"Invalid position"<<endl; return false;}
  nodeType<type> *current = first; *newNode; *current = current->link;
  newNode = new nodeType<type>;
  assert(newNode != NULL);
  new node->info = item;
  new node->link = NULL;
  for(int pos; i<count; i++)
  current = current->link;
```

15

University of Bahrain
College of Information Technology
Department of Computer Science
Summer Semester, 2012 – 13

ITCS215 Data Structures

Quiz 4

Name **ID #** **Sec**

(1) [7 Marks] Write a member function **insertLast** to be included in class **circularLinkedList**. The function inserts a new node at the end of the list. Assume that the class contains a private data member ***last***, which points to the last node of the list.

(2) [8 Marks] Write a member function called **reverseNodes** to be included in class **doublyLinkedList,** which reverses the nodes of the existing list. Do not allocate any new memory space, use the nodes of the existing list. Write the function by swapping the "info" field of the nodes.

```
template <class Type>
void circularLinkedList<Type>::insertLast(const Type &item)
{
    nodeType<Type> *newNode;
    newNode = new nodeType<Type>;
    assert(newNode != NULL);
    newNode->info = item; newNode->link = NULL;
    if (last == NULL){last = newNode; last->link = last;}
    else{
        newNode->link = last->link;
        last->link = newNode;
        last = newNode;}
    count++;
}
```

7

15

University of Bahrain
College of Information Technology
Department of Computer Science
Summer Semester, 2012 – 13

ITCS215 Data Structures

Quiz 5

Name **ID #** **Sec**

(1) [10 Marks] Write a function that accepts two stacks *s1* and *s2*, consisting of integers, as parameters. The function rearranges the elements of the stacks in such a way that stack *s1* contains only positive integers and stack *s2* contains only negative integers. Any zeroes are not stored in *s1* or *s2*. Use common stack operations only.
Function prototype:
void rearrangeStacks(stackType<int> &s1, stackType<int> &s2);

(2) [5 Marks]) Convert the following infix expression to equivalent postfix notation:
((A + B) * (C – D) + E) / (A – B))
AB + CD- *E + AB -/ ✓ 5

```
void rearrangeStacks(stackType<int>& s1, stackType<int>& s2)
{  stackType<int> s3,s4; int item;
   if (s1.isEmptyStack() && s2.isEmptyStack())
      cout<<"Both lists are Empty"<<endl;
   else {  while (!s1.isEmptyStack())
           {  item=s1.top();
              if (item>0) s3.push(item);
              if (item<0) s4.push(item);
              s1.pop();
           }
           while (!s2.isEmptyStack())
           {  item=s2.top();
              if (item>0) s3.push(item);
              if (item<0) s4.push(item);
              s2.pop();
           }
           while (!s3.isEmptyStack())
           {  s1.push(s3.top());
              s3.pop();}
           while (!s4.isEmptyStack())
           {  s2.push(s4.top());
              s4.pop();}
   }
}
```

10